# 前言

这个文档的内容**纯手打**，如果想要看更多的干货文章，关注我的公众号：**Java3y**。有更多的原创技术文章和干货！

目前疯狂处于**疯狂**更新PDF中，只要是Java后端的知识，都会有！**欢迎来我公众号催更！**微信搜索：**Java3y**

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！公众号有我的**联系方式**

- 🔥Java精美脑图
- 🔥Java学习路线
- 🔥开发常用工具
- 🔥精美原创电子书

在公众号下回复「**888**」即可获取！！

**学习不能盲目，跟着我，会让你事半功倍**



电子书的整理也是挺不容易，如果你觉得有帮助，想要打赏作者，那么可以通过这个收款码打赏我，**金额不重要，心意最重要**。主要是我可以通过这个打赏情况来预计大家对这本电子书的评价，嘻嘻


# JDBC入门

# 1.什么是JDBC

JDBC全称为：Java Data Base Connectivity,它是可以执行SQL语句的Java API

# 2.为什么我们要用JDBC

- 市面上有非常多的数据库，本来我们是需要根据不同的数据库学习不同的API，sun公司为了简化这个操作，定义了JDBC API【接口】
- sun公司只是提供了JDBC API【接口】，数据库厂商负责实现。
- 对于我们来说，**操作数据库都是在JDBC API【接口】上**，使用不同的数据库，只要用数据库厂商提供的数据库驱动程序即可
- 这大大简化了我们的学习成本

# 3.简单操作JDBC

步骤:

1. 导入MySQL或者Oracle驱动包
2. 装载数据库驱动程序
3. 获取到与数据库连接
4. 获取可以执行SQL语句的对象
5. 执行SQL语句
6. 关闭连接

```
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {

            /*
            * 加载驱动有两种方式
            *
            * 1：会导致驱动会注册两次，过度依赖于mysql的api，脱离的mysql的开发包，程序则无法编译
            * 2：驱动只会加载一次，不需要依赖具体的驱动，灵活性高
            *
            * 我们一般都是使用第二种方式
            * */

            //1.
            //DriverManager.registerDriver(new com.mysql.jdbc.Driver());

            //2.
            Class.forName("com.mysql.jdbc.Driver");
```

```
            //获取与数据库连接的对象-Connetcion
            connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/zhongfucheng",
"root", "root");

            //获取执行sql语句的statement对象
            statement = connection.createStatement();

            //执行sql语句,拿到结果集
            resultSet = statement.executeQuery("SELECT * FROM users");

            //遍历结果集，得到数据
            while (resultSet.next()) {

                System.out.println(resultSet.getString(1));

                System.out.println(resultSet.getString(2));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } finally {

            /*
            * 关闭资源，后调用的先关闭
            *
            * 关闭之前，要判断对象是否存在
            * */

            if (resultSet != null) {
                try {
                    resultSet.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }

            }
            if (statement != null) {
                try {
                    statement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }

            }
            if (connection != null) {
                try {
```

```
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }

        }

    }
```

上面我们已经简单使用JDBC去查询数据库的数据了，接下来我们去了解一下上面代码用到的对象

---

# 4.Connection对象

**客户端与数据库所有的交互都是通过Connection来完成的。**

常用的方法：

```
//创建向数据库发送sql的statement对象。

createcreateStatement()

//创建向数据库发送预编译sql的PrepareSatement对象。

prepareStatement(sql)

//创建执行存储过程的callableStatement对象

prepareCall(sql)

//设置事务自动提交

setAutoCommit(boolean autoCommit)

//提交事务

commit()

//回滚事务

rollback()
```


如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

---

# 5.Statement对象

**Statement对象用于向数据库发送Sql语句**，对数据库的增删改查都可以通过此对象发送sql语句完成。

Statement对象的常用方法：

```
//查询

executeQuery(String sql)

//增删改

executeUpdate(String sql)

//任意sql语句都可以，但是目标不明确，很少用

execute(String sql)
```

```
//把多条的sql语句放进同一个批处理中

addBatch(String sql)

//向数据库发送一批sql语句执行

executeBatch()
```

# 6.ResultSet对象

**ResultSet对象代表Sql语句的执行结果**，当Statement对象执行executeQuery()时，会返回一个ResultSet对象

ResultSet对象维护了一个数据行的游标【简单理解成指针】，调用ResultSet.next()方法，可以让游标指向具体的数据行，进行获取该行的数据

常用方法：

```
//获取任意类型的数据

getObject(String columnName)

//获取指定类型的数据【各种类型，查看API】

getString(String columnName)

//对结果集进行滚动查看的方法

next()

Previous()

absolute(int row)

beforeFirst()

afterLast()
```

# 7.写一个简单工具类

通过上面的理解，我们已经能够使用JDBC对数据库的数据进行增删改查了，我们发现，无论增删改查都需要连接数据库，关闭资源，所以我们**把连接数据库，释放资源的操作抽取到一个工具类**

```
    /*
    * 连接数据库的driver，url，username，password通过配置文件来配置，可以增加灵活性
    * 当我们需要切换数据库的时候，只需要在配置文件中改以上的信息即可
    *
    * */

    private static String  driver = null;
    private static String  url = null;
    private static String  username = null;
    private static String password = null;

    static {
        try {

            //获取配置文件的读入流
            InputStream inputStream =
UtilsDemo.class.getClassLoader().getResourceAsStream("db.properties");

            Properties properties = new Properties();
            properties.load(inputStream);

            //获取配置文件的信息
            driver = properties.getProperty("driver");
            url = properties.getProperty("url");
            username = properties.getProperty("username");
            password = properties.getProperty("password");

            //加载驱动类
            Class.forName(driver);


        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

    }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url,username,password);
    }
    public static void release(Connection connection, Statement statement,
ResultSet resultSet) {

        if (resultSet != null) {
```

```
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
```

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

# JDBC使用的一些细节

## 1.PreparedStatement对象

**PreparedStatement对象继承Statement对象，它比Statement对象更强大，使用起来更简单**

1. Statement对象编译SQL语句时，如果SQL语句有变量，就需要使用分隔符来隔开，如果变量非常多，就会使SQL变得非常复杂。**PreparedStatement可以使用占位符，简化sql的编写**
2. Statement会频繁编译SQL。**PreparedStatement可对SQL进行预编译，提高效率，预编译的SQL存储在PreparedStatement对象中**
3. **PreparedStatement防止SQL注入。**【Statement通过分隔符'++',编写永等式，可以不需要密码就进入数据库】

```
        //模拟查询id为2的信息
        String id = "2";

        Connection connection = UtilsDemo.getConnection();

        String sql = "SELECT * FROM users WHERE id = ?";
        PreparedStatement preparedStatement =
connection.preparedStatement(sql);

        //第一个参数表示第几个占位符【也就是?号】，第二个参数表示值是多少
        preparedStatement.setString(1,id);

        ResultSet resultSet = preparedStatement.executeQuery();
```

```
        if (resultSet.next()) {
            System.out.println(resultSet.getString("name"));
        }

        //释放资源
        UtilsDemo.release(connection, preparedStatement, resultSet);
```

---

# 2.批处理

当需要向数据库发送一批SQL语句执行时，应避免向数据库一条条发送执行，**采用批处理以提升执行效率**

批处理有两种方式：

1. Statement
2. PreparedStatement

**通过executeBath()方法批量处理执行SQL语句，返回一个int[]数组，该数组代表各句SQL的返回值**

以下代码是以Statement方式实现批处理

```
        /*
        * Statement执行批处理
        *
        * 优点：
        *       可以向数据库发送不同的SQL语句
        * 缺点：
        *       SQL没有预编译
        *       仅参数不同的SQL，需要重复写多条SQL
        * */
        Connection connection = UtilsDemo.getConnection();

        Statement statement = connection.createStatement();
        String sql1 = "UPDATE users SET name='zhongfucheng' WHERE id='3'";
        String sql2 = "INSERT INTO users (id, name, password, email,
birthday)" +
                " VALUES('5','nihao','123','ss@qq.com','1995-12-1')";

        //将sql添加到批处理
        statement.addBatch(sql1);
        statement.addBatch(sql2);

        //执行批处理
        statement.executeBatch();
```

```
        //清空批处理的sql
        statement.clearBatch();

        UtilsDemo.release(connection, statement, null);
```

以下方式以PreparedStatement方式实现批处理

```
        /*
        * PreparedStatement批处理
        *   优点:
        *       SQL语句预编译了
        *       对于同一种类型的SQL语句，不用编写很多条
        *   缺点:
        *       不能发送不同类型的SQL语句
        *
        * */
        Connection connection = UtilsDemo.getConnection();

        String sql = "INSERT INTO test(id,name) VALUES (?,?)";
        PreparedStatement preparedStatement =
connection.prepareStatement(sql);

        for (int i = 1; i <= 205; i++) {
            preparedStatement.setInt(1, i);
            preparedStatement.setString(2, (i + "zhongfucheng"));

            //添加到批处理中
            preparedStatement.addBatch();

            if (i %2 ==100) {

                //执行批处理
                preparedStatement.executeBatch();

                //清空批处理【如果数据量太大，所有数据存入批处理，内存肯定溢出】
                preparedStatement.clearBatch();
            }

        }
        //不是所有的%2==100，剩下的再执行一次批处理
        preparedStatement.executeBatch();

        //再清空
        preparedStatement.clearBatch();

        UtilsDemo.release(connection, preparedStatement, null);
```

# 3.处理大文本和二进制数据

**clob和blob**

- clob用于存储大文本
- blob用于存储二进制数据

## 3.1 MYSQL

**MySQL存储大文本是用Test【代替clob】**，Test又分为4类

- TINYTEXT
- TEXT
- MEDIUMTEXT
- LONGTEXT

**同理blob也有这4类**

下面用JDBC连接MySQL数据库去操作大文本数据和二进制数据

```
/*
*用JDBC操作MySQL数据库去操作大文本数据
*
*setCharacterStream(int parameterIndex,java.io.Reader reader,long length)
*第二个参数接收的是一个流对象，因为大文本不应该用String来接收，String太大会导致内存溢出
*第三个参数接收的是文件的大小
*
* */
public class Demo5 {

    @Test
    public void add() {

        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
```

```
        try {
            connection = JdbcUtils.getConnection();
            String sql = "INSERT INTO test2 (bigTest) VALUES(?) ";
            preparedStatement = connection.prepareStatement(sql);

            //获取到文件的路径
            String path =
Demo5.class.getClassLoader().getResource("BigTest").getPath();
            File file = new File(path);
            FileReader fileReader = new FileReader(file);

            //第三个参数，由于测试的Mysql版本过低，所以只能用int类型的。高版本的不需要进行
强转
            preparedStatement.setCharacterStream(1, fileReader, (int)
file.length());

            if (preparedStatement.executeUpdate() > 0) {
                System.out.println("插入成功");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } finally {
            JdbcUtils.release(connection, preparedStatement, null);
        }


    }

    /*
    * 读取大文本数据，通过ResultSet中的getCharacterStream()获取流对象数据
    *
    * */
    @Test
    public void read() {

        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = JdbcUtils.getConnection();
            String sql = "SELECT * FROM test2";
            preparedStatement = connection.prepareStatement(sql);
            resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
```

```
                Reader reader = resultSet.getCharacterStream("bigTest");

                FileWriter fileWriter = new FileWriter("d:\\abc.txt");
                char[] chars = new char[1024];
                int len = 0;
                while ((len = reader.read(chars)) != -1) {
                    fileWriter.write(chars, 0, len);
                    fileWriter.flush();
                }
                fileWriter.close();
                reader.close();

            }
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            JdbcUtils.release(connection, preparedStatement, resultSet);
        }

    }
```

```
/*
* 使用JDBC连接MYsql数据库操作二进制数据
* 如果我们要用数据库存储一个大视频的时候，数据库是存储不到的。
* 需要设置max_allowed_packet，一般我们不使用数据库去存储一个视频
* */
public class Demo6 {

    @Test
    public void add() {


        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;


        try {
            connection = JdbcUtils.getConnection();
            String sql = "INSERT INTO test3 (blobtest) VALUES(?)";
            preparedStatement = connection.prepareStatement(sql);
```

```
            //获取文件的路径和文件对象
            String path =
Demo6.class.getClassLoader().getResource("1.wmv").getPath();
            File file = new File(path);

            //调用方法
            preparedStatement.setBinaryStream(1, new FileInputStream(path),
(int)file.length());

            if (preparedStatement.executeUpdate() > 0) {

                System.out.println("添加成功");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } finally {
            JdbcUtils.release(connection, preparedStatement, null);
        }

    }

    @Test
    public void read() {


        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;


        try {
            connection = JdbcUtils.getConnection();
            String sql = "SELECT * FROM test3";
            preparedStatement = connection.prepareStatement(sql);

            resultSet = preparedStatement.executeQuery();


            //如果读取到数据，就把数据写到磁盘下
            if (resultSet.next()) {
                InputStream inputStream =
resultSet.getBinaryStream("blobtest");
                FileOutputStream fileOutputStream = new
FileOutputStream("d:\\aa.jpg");

                int len = 0;
```

```
                byte[] bytes = new byte[1024];
                while ((len = inputStream.read(bytes)) > 0) {

                    fileOutputStream.write(bytes, 0, len);

                }
                fileOutputStream.close();
                inputStream.close();

            }

        } catch (SQLException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            JdbcUtils.release(connection, preparedStatement, null);
        }

    }
```

## 3.2 Oracle

下面用JDBC连接Oracle数据库去操作大文本数据和二进制数据

```
//使用JDBC连接Oracle数据库操作二进制数据

/*
* 对于Oracle数据库和Mysql数据库是有所不同的。
* 1.Oracle定义了BLOB字段，但是这个字段不是真正地存储二进制数据
* 2.向这个字段存一个BLOB指针，获取到Oracle的BLOB对象,把二进制数据放到这个指针里面,指针指向BLOB字段
* 3.需要事务支持
*
* */
public class Demo7 {
    @Test
    public void add() {


        Connection connection = null;
        PreparedStatement preparedStatement = null;
```

```
        ResultSet resultSet = null;

        try {
            connection = UtilsDemo.getConnection();

            //开启事务
            connection.setAutoCommit(false);

            //插入一个BLOB指针
            String sql = "insert into test4(id,image) values(?,empty_blob())";
            preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setInt(1, 1);
            preparedStatement.executeUpdate();

            //把BLOB指针查询出来,得到BLOB对象
            String sql2 = "select image from test4 where id= ? for update";
            preparedStatement = connection.prepareStatement(sql2);
            preparedStatement.setInt(1, 1);
            resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                //得到Blob对象--当成是Oracle的Blob,不是JDBC的,所以要强转[导的是
oracle.sql.BLOB包]
                BLOB  blob = (BLOB) resultSet.getBlob("image");

                //写入二进制数据
                OutputStream outputStream = blob.getBinaryOutputStream();

                //获取到读取文件读入流
                InputStream inputStream =
Demo7.class.getClassLoader().getResourceAsStream("01.jpg");

                int len=0;
                byte[] bytes = new byte[1024];
                while ((len = inputStream.read(bytes)) > 0) {

                    outputStream.write(bytes, 0, len);
                }
                outputStream.close();
                inputStream.close();
        connection.setAutoCommit(true);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
```

```
            UtilsDemo.release(connection, preparedStatement, null);
        }

    }

    @Test
    public void find() {

        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {
            connection = UtilsDemo.getConnection();
            String sql = "SELECT * FROM test4 WHERE id=1";

            preparedStatement = connection.prepareStatement(sql);
            resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {

                //获取到BLOB对象
                BLOB blob = (BLOB) resultSet.getBlob("image");

                //将数据读取到磁盘上
                InputStream inputStream = blob.getBinaryStream();
                FileOutputStream fileOutputStream = new
FileOutputStream("d:\\zhongfucheng.jpg");
                int len=0;
                byte[] bytes = new byte[1024];

                while ((len = inputStream.read(bytes)) > 0) {

                    fileOutputStream.write(bytes, 0, len);
                }

                inputStream.close();
                fileOutputStream.close();

            }

        } catch (SQLException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            UtilsDemo.release(connection, preparedStatement, null);
```

```
            }
        }
    }
```

对于JDBC连接Oracle数据库操作CLOB数据,我就不再重复了,操作跟BLOB几乎相同


如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

---

# 4.获取数据库的自动主键列

## 4.1 为什么要获取数据库的自动主键列数据?

应用场景:

有一张老师表，一张学生表。**现在来了一个新的老师，学生要跟着新老师上课。**

我首先要知道老师的id编号是多少，学生才能知道跟着哪个老师学习**【学生外键参照老师主键】**。

```
@Test
public void test() {

    Connection connection = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;

    try {
        connection = JdbcUtils.getConnection();

        String sql = "INSERT INTO test(name) VALUES(?)";
        preparedStatement = connection.prepareStatement(sql);

        preparedStatement.setString(1, "ouzicheng");

        if (preparedStatement.executeUpdate() > 0) {

            //获取到自动主键列的值
            resultSet = preparedStatement.getGeneratedKeys();

            if (resultSet.next()) {
                int id = resultSet.getInt(1);
                System.out.println(id);
            }
        }

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        JdbcUtils.release(connection, preparedStatement, null);
    }
```

# 5.调用数据库的存储过程

**调用存储过程的语法：**

```
{call <procedure-name>[(<arg1>,<arg2>, ...)]}
```

**调用函数的语法：**

```
  {?= call <procedure-name>[(<arg1>,<arg2>, ...)]}
```

**如果是Output类型的，那么在JDBC调用的时候是要注册的。**如下代码所示：

```
/*
    jdbc调用存储过程

  delimiter $$
    CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), INOUT inOutParam
varchar(255))
    BEGIN
        SELECT CONCAT('zyxw---', inputParam) into inOutParam;
    END $$
  delimiter ;
*/
//我们在JDBC调用存储过程,就像在调用方法一样
public class Demo9 {

    public static void main(String[] args) {
        Connection connection = null;
        CallableStatement callableStatement = null;

        try {
            connection = JdbcUtils.getConnection();

            callableStatement = connection.prepareCall("{call demoSp(?,?)}");

            callableStatement.setString(1, "nihaoa");

            //注册第2个参数,类型是VARCHAR
            callableStatement.registerOutParameter(2, Types.VARCHAR);
            callableStatement.execute();

      //获取传出参数[获取存储过程里的值]
            String result = callableStatement.getString(2);
            System.out.println(result);

        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            try {
                connection.close();
                callableStatement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

```
    }
```

参考资料：

```
--------------------------------------------------------------------------------
----过程

#修改mysql语句的结果符为//
mysql > delimiter //

#定义一个过程，获取users表总记录数，将10设置到变量count中
create procedure simpleproc(out count int)
begin
    select count(id) into count from users;
end
//

#修改mysql语句的结果符为;
mysql > delimiter ;

#调用过程，将结果覆给变量a，@是定义变量的符号
call simpleproc(@a);

#显示变量a的值
select @a;

//以下是Java调用Mysql的过程
  String sql = "{call simpleproc(?)}";
  Connection conn = JdbcUtil.getConnection();
  CallableStatement cstmt = conn.prepareCall(sql);
  cstmt.registerOutParameter(1,Types.INTEGER);
  cstmt.execute();
  Integer count = cstmt.getInt(1);
  System.out.println("共有" + count + "人");

--------------------------------------------------------------------------------
----函数

#修改mysql语句的结果符为//
mysql > delimiter //

#定义一个函数，完成字符串拼接
create function hello( s char(20) ) returns char(50)
return concat('hello, ',s,'!');
//
```

```
#修改mysql语句的结果符为;
mysql > delimiter ;

#调用函数
select hello('world');

//以下是Java调用Mysql的函数
  String sql = "{? = call hello(?)}";
  Connection conn = JdbcUtil.getConnection();
  CallableStatement cstmt = conn.prepareCall(sql);
  cstmt.registerOutParameter(1,Types.VARCHAR);
  cstmt.setString(2,"zhaojun");
  cstmt.execute();
  String value = cstmt.getString(1);
  System.out.println(value);
  JdbcUtil.close(cstmt);
  JdbcUtil.close(conn);
```

加油

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

# 事务+元数据+改造工具类

## 1.事务

> 一个SESSION所进行的所有更新操作要么一起成功，要么一起失败

举个例子:**A向B转账，转账这个流程中如果出现问题，事务可以让数据恢复成原来一样【A账户的钱没变，B账户的钱也没变】。**

**事例说明：**

```
/*
* 我们来模拟A向B账号转账的场景
*    A和B账户都有1000块，现在我让A账户向B账号转500块钱
*
* */
        //JDBC默认的情况下是关闭事务的，下面我们看看关闭事务去操作转账操作有什么问题

        //A账户减去500块
        String sql = "UPDATE a SET money=money-500 ";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.executeUpdate();

        //B账户多了500块
        String sql2 = "UPDATE b SET money=money+500";
        preparedStatement = connection.prepareStatement(sql2);
        preparedStatement.executeUpdate();
```

从上面看，我们的确可以发现A向B转账，成功了。可是**如果A向B转账的过程中出现了问题呢？**下面模拟一下

```
        //A账户减去500块
              String sql = "UPDATE a SET money=money-500 ";
              preparedStatement = connection.prepareStatement(sql);
              preparedStatement.executeUpdate();

        //这里模拟出现问题
              int a = 3 / 0;


              String sql2 = "UPDATE b SET money=money+500";
              preparedStatement = connection.prepareStatement(sql2);
              preparedStatement.executeUpdate();
```

显然，上面**代码是会抛出异常的**，我们再来查询一下数据。**A账户少了500块钱，B账户的钱没有增加。这明显是不合理的。**

我们可以通过事务来解决上面出现的问题

```
        //开启事务,对数据的操作就不会立即生效。
              connection.setAutoCommit(false);

              //A账户减去500块
              String sql = "UPDATE a SET money=money-500 ";
              preparedStatement = connection.prepareStatement(sql);
              preparedStatement.executeUpdate();

              //在转账过程中出现问题
              int a = 3 / 0;

              //B账户多500块
              String sql2 = "UPDATE b SET money=money+500";
              preparedStatement = connection.prepareStatement(sql2);
              preparedStatement.executeUpdate();

              //如果程序能执行到这里，没有抛出异常，我们就提交数据
              connection.commit();

        //关闭事务【自动提交】
        connection.setAutoCommit(true);


          } catch (SQLException e) {
              try {
                  //如果出现了异常，就会进到这里来，我们就把事务回滚【将数据变成原来那样】
                  connection.rollback();
```

```
            //关闭事务【自动提交】
            connection.setAutoCommit(true);
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
```

上面的程序也一样抛出了异常，A账户钱没有减少，B账户的钱也没有增加。

注意：当Connection遇到一个未处理的SQLException时，系统会非正常退出，事务也会自动回滚，但**如果程序捕获到了异常，是需要在catch中显式回滚事务的。**

## 1.1 savapoint

我们还可以使用savepoint设置中间点。如果在某地方出错了，我们设置中间点，回滚到出错之前即可。

应用场景：现在我们要算一道数学题，算到后面发现算错数了。前面的运算都是正确的，我们不可能重头再算【直接rollback】，最好的做法就是在**保证前面算对的情况下，设置一个保存点。从保存点开始重新算。**

注意：**savepoint不会结束当前事务，普通提交和回滚都会结束当前事务的**

---

## 1.2 事务的隔离级别

数据库定义了4个隔离级别：

1. Serializable【可避免脏读，不可重复读，虚读】
2. Repeatable read【可避免脏读，不可重复读】
3. Read committed【可避免脏读】
4. Read uncommitted【级别最低，什么都避免不了】

分别对应Connection类中的4个常量

1. **TRANSACTION_READ_UNCOMMITTED**
2. **TRANSACTION_READ_COMMITTED**
3. **TRANSACTION_REPEATABLE_READ**
4. **TRANSACTION_SERIALIZABLE**

---

脏读：**一个事务读取到另外一个事务未提交的数据**

例子：A向B转账，**A执行了转账语句，但A还没有提交事务，B读取数据，发现自己账户钱变多了**！B跟A说，我已经收到钱了。A回滚事务【rollback】，等B再查看账户的钱时，发现钱并没有多。

---

不可重复读：**一个事务读取到另外一个事务已经提交的数据，也就是说一个事务可以看到其他事务所做的修改**

注：**A查询数据库得到数据，B去修改数据库的数据，导致A多次查询数据库的结果都不一样【危害：A每次查询的结果都是受B的影响的，那么A查询出来的信息就没有意思了】**

---

虚读(幻读)：**是指在一个事务内读取到了别的事务插入的数据，导致前后读取不一致。**

注：**和不可重复读类似，但虚读(幻读)会读到其他事务的插入的数据，导致前后读取不一致**

---

简单总结：**脏读是不可容忍的，不可重复读和虚读在一定的情况下是可以的【做统计的肯定就不行】**。

---

# 2.元数据

## 2.1 什么是元数据

元数据其实就是数据库，表，列的定义信息

## 2.2 为什么我们要用元数据

即使我们写了一个简单工具类，我们的代码还是非常冗余。**对于增删改而言，只有SQL和参数是不同的**，我们为何不把这些相同的代码抽取成一个方法？对于**查询而言，不同的实体查询出来的结果集是不一样的**。我们要**使用元数据获取结果集的信息，才能对结果集进行操作**。

- ParameterMetaData　--参数的元数据
- ResultSetMetaData　--结果集的元数据
- DataBaseMetaData　--数据库的元数据

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

# 3.改造JDBC工具类

问题：我们**对数据库的增删改查都要连接数据库，关闭资源，获取PreparedSteatment对象，获取Connection对象此类的操作，这样的代码重复率是极高的**，所以我们要对工具类进行增强

## 3.1 增删改

```
    //我们发现，增删改只有SQL语句和传入的参数是不知道的而已，所以让调用该方法的人传递进来

  //由于传递进来的参数是各种类型的，而且数目是不确定的，所以使用Object[]

    public static void update(String sql, Object[] objects) {

        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {
            connection = getConnection();
            preparedStatement = connection.prepareStatement(sql);

            //根据传递进来的参数，设置SQL占位符的值
            for (int i = 0; i < objects.length; i++) {
                preparedStatement.setObject(i + 1, objects[i]);
            }

            //执行SQL语句
            preparedStatement.executeUpdate();
```

```
        } catch (Exception e) {
            e.printStackTrace();
```

## 3.2 查询

```
    /*
        1:对于查询语句来说，我们不知道对结果集进行什么操作【常用的就是把数据封装成一个Bean对象，封装成一个List集合】
        2:我们可以定义一个接口，让调用者把接口的实现类传递进来
        3:这样接口调用的方法就是调用者传递进来实现类的方法。【策略模式】

    */
    //这个方法的返回值是任意类型的，所以定义为Object。
    public static Object query(String sql, Object[] objects, ResultSetHandler rsh) {

        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;

        try {
            connection = getConnection();
            preparedStatement = connection.prepareStatement(sql);

            //根据传递进来的参数，设置SQL占位符的值
            if (objects != null) {
                for (int i = 0; i < objects.length; i++) {
                    preparedStatement.setObject(i + 1, objects[i]);
                }
            }


            resultSet = preparedStatement.executeQuery();

            //调用调用者传递进来实现类的方法，对结果集进行操作
            return rsh.hanlder(resultSet);
  }
```

**接口：**

```
/*
* 定义对结果集操作的接口，调用者想要对结果集进行什么操作，只要实现这个接口即可
* */
public interface ResultSetHandler {
     Object hanlder(ResultSet resultSet);

}
```

**实现类：**

```
//接口实现类，对结果集封装成一个Bean对象
public class BeanHandler implements ResultSetHandler {


    //要封装成一个Bean对象，首先要知道Bean是什么，这个也是调用者传递进来的。
    private Class clazz;

    public BeanHandler(Class clazz) {
        this.clazz = clazz;
    }

    @Override
    public Object hanlder(ResultSet resultSet) {

        try {

            //创建传进对象的实例化
            Object bean = clazz.newInstance();

            if (resultSet.next()) {

                //拿到结果集元数据
                ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

                for (int i = 0; i < resultSetMetaData.getColumnCount(); i++) {

                    //获取到每列的列名
                    String columnName = resultSetMetaData.getColumnName(i+1);

                    //获取到每列的数据
                    String columnData = resultSet.getString(i+1);

                    //设置Bean属性
                    Field field = clazz.getDeclaredField(columnName);
```

```
                field.setAccessible(true);
                field.set(bean,columnData);
            }

            //返回Bean对象
            return bean;
        }
```

---

【策略模式】简单理解：

- 我们并不知道调用者想对结果集进行怎么样的操作，于是**让调用者把想要做的操作对象传递过来**
- 我们只要**用传递过来的对象对结果集进行封装**就好了。
    - 至于调用者会传递什么对象过来，该对象要实现什么方法。我们可以**使用接口来对其规范**


如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

# 数据库连接池+DBUtils+分页

## 1.数据库连接池

### 1.1什么是数据库连接池

简单来说：**数据库连接池就是提供连接的。。。**

### 1.2为什么我们要使用数据库连接池

- 数据库的连接的建立和关闭是非常消耗资源的
- 频繁地打开、关闭连接造成系统性能低下

### 1.3如何自己编写一个连接池

1. 编写连接池需**实现java.sql.DataSource接口**
2. **创建批量的Connection用LinkedList保存**【既然是个池，当然用集合保存、、LinkedList底层是链表，对增删性能较好】
3. **实现getConnetion()**，让getConnection()每次调用，都是**在LinkedList中取一个Connection返回给用户**
4. **调用Connection.close()方法，Connction返回给LinkedList**

```
    private static LinkedList<Connection> list = new LinkedList<>();

    //获取连接只需要一次就够了，所以用static代码块
    static {
        //读取文件配置
        InputStream inputStream =
Demo1.class.getClassLoader().getResourceAsStream("db.properties");

        Properties properties = new Properties();
        try {
            properties.load(inputStream);
            String url = properties.getProperty("url");
            String username = properties.getProperty("username");
            String driver = properties.getProperty("driver");
            String password = properties.getProperty("password");

            //加载驱动
            Class.forName(driver);

            //获取多个连接，保存在LinkedList集合中
            for (int i = 0; i < 10; i++) {
                Connection connection = DriverManager.getConnection(url,
username, password);
```

```
                list.add(connection);
            }

        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }

    //重写Connection方法，用户获取连接应该从LinkedList中给他
    @Override
    public Connection getConnection() throws SQLException {
        System.out.println(list.size());
        System.out.println(list);

       //先判断LinkedList是否存在连接
       return list.size() > 0 ? list.removeFirst() : null;
    }
```

我们已经完成前三步了，现在问题来了。**我们调用Conncetion.close()方法，是把数据库的物理连接关掉，而不是返回给LinkedList的**

解决思路：

1. 写一个Connection子类，覆盖close()方法
2. 写一个Connection包装类，增强close()方法
3. 用动态代理，返回一个代理对象出去，拦截close()方法的调用，对close()增强

分析第一个思路：

- **Connection是通过数据库驱动加载的，保存了数据的信息**。写一个子类Connection，new出对象，**子类的Connction无法直接继承父类的数据信息，也就是说子类的Connection是无法连接数据库的**，更别谈覆盖close()方法了。

分析第二个思路：

- 写一个Connection包装类。
    1. 写一个类，实现与被增强对象的相同接口【Connection接口】
    2. 定义一个变量，指向被增强的对象
    3. 定义构造方法，接收被增强对象
    4. 覆盖想增强的方法
    5. 对于不想增强的方法，直接调用被增强对象的方法
- 这个思路本身是没什么毛病的，就是**实现接口时，方法太多了！**，所以我们也不使用此方法

分析第三个思路代码实现：

```
    @Override
    public Connection getConnection() throws SQLException {

        if (list.size() > 0) {
            final Connection connection = list.removeFirst();

            //看看池的大小
            System.out.println(list.size());

            //返回一个动态代理对象
            return (Connection)
Proxy.newProxyInstance(Demo1.class.getClassLoader(),
connection.getClass().getInterfaces(), new InvocationHandler() {

                @Override
                public Object invoke(Object proxy, Method method, Object[]
args) throws Throwable {

                    //如果不是调用close方法，就按照正常的来调用
                    if (!method.getName().equals("close")) {
                        method.invoke(connection, args);
                    } else {

                        //进到这里来，说明调用的是close方法
                        list.add(connection);

                        //再看看池的大小
                        System.out.println(list.size());

                    }
                    return null;
                }

            });
        }
        return null;
    }
```

---

我们上面已经能够简单编写一个线程池了。下面我们来使用一下开源数据库连接池

## 1.4DBCP

使用DBCP数据源的步骤：

1. 导入两个jar包【Commons-dbcp.jar和Commons-pool.jar】
2. 读取配置文件
3. 获取BasicDataSourceFactory对象
4. 创建DataSource对象

```
    private static DataSource dataSource = null;

    static {
        try {
            //读取配置文件
            InputStream inputStream =
Demo3.class.getClassLoader().getResourceAsStream("dbcpconfig.properties");
            Properties properties = new Properties();
            properties.load(inputStream);

            //获取工厂对象
            BasicDataSourceFactory basicDataSourceFactory = new
BasicDataSourceFactory();
            dataSource = basicDataSourceFactory.createDataSource(properties);

        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
        return dataSource.getConnection();

    }

    //这里释放资源不是把数据库的物理连接释放了，是把连接归还给连接池【连接池的Connection内
部自己做好了】
    public static void release(Connection conn, Statement st, ResultSet rs) {

        if (rs != null) {
            try {
                rs.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            rs = null;
        }
        if (st != null) {
            try {
                st.close();
            } catch (Exception e) {
                e.printStackTrace();
```

```
            }

        }
        if (conn != null) {
            try {
                conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }

        }
    }
```

## 1.5 C3P0

C3P0数据源的性能更胜一筹，并且它可以**使用XML配置文件配置信息！**

步骤：

1. 导入开发包【c3p0-0.9.2-pre1.jar】和【mchange-commons-0.2.jar】
2. 导入XML配置文件【可以在程序中自己一个一个配，C3P0的doc中的Configuration有XML文件的事例】
3. new出ComboPooledDataSource对象

```
    private static ComboPooledDataSource comboPooledDataSource = null;

    static {
        //如果我什么都不指定，就是使用XML默认的配置，这里我指定的是oracle的
        comboPooledDataSource = new ComboPooledDataSource("oracle");
    }

    public static Connection getConnection() throws SQLException {
        return comboPooledDataSource.getConnection();
    }
```

---

## 1.6 Tomcat数据源

Tomcat服务器也给我们提供了连接池，内部其实就是DBCP

步骤：

1. 在META-INF目录下配置context.xml文件【文件内容可以在tomcat默认页面的 JNDI Resources下Configure Tomcat's Resource Factory找到】
2. 导入Mysql或oracle开发包到tomcat的lib目录下
3. 初始化JNDI->获取JNDI容器->检索以XXX为名字在JNDI容器存放的连接池

context.xml文件的配置：

```
<Context>

  <Resource name="jdbc/EmployeeDB"
            auth="Container"
            type="javax.sql.DataSource"

            username="root"
            password="root"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/zhongfucheng"
            maxActive="8"
            maxIdle="4"/>
</Context>
```

```
        try {

      //初始化JNDI容器
            Context initCtx = new InitialContext();

      //获取到JNDI容器
            Context envCtx = (Context) initCtx.lookup("java:comp/env");

      //扫描以jdbc/EmployeeDB名字绑定在JNDI容器下的连接池
            DataSource ds = (DataSource)
                    envCtx.lookup("jdbc/EmployeeDB");

            Connection conn = ds.getConnection();
            System.out.println(conn);

        }
```

## 1.7 Druid

可以看看文档，现在这个数据库连接池用得挺多的：https://github.com/alibaba/druid


如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

# 2. 使用dbutils框架

**dbutils它是对JDBC的简单封装，极大简化jdbc编码的工作量**

## 2.1DbUtils类

提供了**关闭连接，装载JDBC驱动，回滚提交事务等方法**的工具类【比较少使用，因为我们学了连接池，就应该使用连接池连接数据库】

## 2.2QueryRunner类

该类**简化了SQL查询，配合ResultSetHandler使用，可以完成大部分的数据库操作**，重载了许多的查询，更新，批处理方法。大大减少了代码量

## 2.3ResultSetHandler接口

该接口**规范了对ResultSet的操作**，要对结果集进行什么操作，传入ResultSetHandler接口的实现类即可。

- ArrayHandler：把结果集中的第一行数据转成对象数组。
- ArrayListHandler：把结果集中的每一行数据都转成一个数组，再存放到List中。
- BeanHandler：将结果集中的第一行数据封装到一个对应的JavaBean实例中。
- BeanListHandler：将结果集中的每一行数据都封装到一个对应的JavaBean实例中，存放到List里。
- ColumnListHandler：将结果集中某一列的数据存放到List中。
- KeyedHandler(name)：将结果集中的每一行数据都封装到一个Map里，再把这些map再存到一个map里，其key为指定的key。
- MapHandler：将结果集中的第一行数据封装到一个Map里，key是列名，value就是对应的值。
- MapListHandler：将结果集中的每一行数据都封装到一个Map里，然后再存放到List
- ScalarHandler 将ResultSet的一个列到一个对象中。

使用DbUtils框架对数据库的CRUD

```
/*
* 使用DbUtils框架对数据库的CRUD
* 批处理
*
* */
public class Test {

    @org.junit.Test
    public void add() throws SQLException {

        //创建出QueryRunner对象
        QueryRunner queryRunner = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "INSERT INTO student (id,name) VALUES(?,?)";

        //我们发现query()方法有的需要传入Connection对象，有的不需要传入
        //区别：你传入Connection对象是需要你来销毁该Connection，你不传入，由程序帮你把Connection放回到连接池中
        queryRunner.update(sql, new Object[]{"100", "zhongfucheng"});

    }

    @org.junit.Test
    public void query()throws SQLException {

        //创建出QueryRunner对象
        QueryRunner queryRunner = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "SELECT * FROM student";

        List list = (List) queryRunner.query(sql, new BeanListHandler(Student.class));
        System.out.println(list.size());
```

```
    }

    @org.junit.Test
    public void delete() throws SQLException {
        //创建出QueryRunner对象
        QueryRunner queryRunner = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "DELETE FROM student WHERE id='100'";

        queryRunner.update(sql);
    }

    @org.junit.Test
    public void update() throws SQLException {
        //创建出QueryRunner对象
        QueryRunner queryRunner = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "UPDATE student SET name=? WHERE id=?";

        queryRunner.update(sql, new Object[]{"zhongfuchengaaa", 1});
    }

    @org.junit.Test
    public void batch() throws SQLException {
        //创建出QueryRunner对象
        QueryRunner queryRunner = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "INSERT INTO student (name,id) VALUES(?,?)";

        Object[][] objects = new Object[10][];
        for (int i = 0; i < 10; i++) {
            objects[i] = new Object[]{"aaa", i + 300};
        }
        queryRunner.batch(sql, objects);
    }

}
```

# 3.分页

分页技术是非常常见的，**在搜索引擎下搜索页面，不可能把全部数据都显示在一个页面里边**。所以我们用到了分页技术。

## 3.1Oracle实现分页

```
  /*
    Oracle分页语法:
      @lineSize---每页显示数据行数
      @currentPage----当前所在页
```

```
*/
SELECT *FROM (
    SELECT 列名,列名,ROWNUM rn
    FROM 表名
    WHERE ROWNUM<=(currentPage*lineSize)) temp

WHERE temp.rn>(currentPage-1)*lineSize;
```

**Oracle分页原理简单解释**:

```
/*
  Oracle分页:
    Oracle的分页依赖于ROWNUM这个伪列，ROWNUM主要作用就是产生行号。

  分页原理:
    1：子查询查出前n行数据，ROWNUM产生前N行的行号
    2：使用子查询产生ROWNUM的行号，通过外部的筛选出想要的数据

  例子:
    我现在规定每页显示5行数据【lineSize=5】，我要查询第2页的数据【currentPage=2】
    注：【对照着语法来看】

  实现:
    1：子查询查出前10条数据【ROWNUM<=10】
    2：外部筛选出后面5条数据【ROWNUM>5】
  3：这样我们就取到了后面5条的数据
*/
```

---

## 3.2Mysql实现分页

```
/*
  Mysql分页语法:
  @start---偏移量，不设置就是从0开始【也就是(currentPage-1)*lineSize】
  @length---长度，取多少行数据

*/
SELECT *
FROM 表名
LIMIT [START], length;

/*
  例子:
    我现在规定每页显示5行数据，我要查询第2页的数据
```

```
    分析：
      1：第2页的数据其实就是从第6条数据开始，取5条

    实现：
      1：start为5【偏移量从0开始】
      2：length为5

*/
```

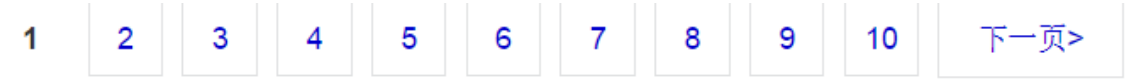
**总结：**

- Mysql从(currentPage-1)*lineSize开始取数据，取lineSize条数据
- Oracle先获取currentPage*lineSize条数据，从(currentPage-1)*lineSize开始取数据

---

## 3.3使用JDBC连接数据库实现分页

下面是常见的分页图片




---

配合图片，看下我们的需求是什么：

1. **算出有多少页的数据，显示在页面上**
2. **根据页码，从数据库显示相对应的数据。**

分析：

1. 算出有多少页数据这是非常简单的【在数据库中查询有多少条记录，你每页显示多少条记录，就可以算出有多少页数据了】
2. 使用Mysql或Oracle的分页语法即可

通过上面分析，我们会发现需要用到4个变量

- currentPage--当前页【由用户决定的】
- totalRecord--总数据数【查询表可知】
- lineSize--每页显示数据的数量【由我们开发人员决定】
- pageCount--页数【totalRecord和lineSize决定】

```
        //每页显示3条数据
        int lineSize = 3;

        //总记录数
        int totalRecord = getTotalRecord();

        //假设用户指定的是第2页
```

```
        int currentPage = 2;

        //一共有多少页
        int pageCount = getPageCount(totalRecord, lineSize);

        //使用什么数据库进行分页，记得要在JdbcUtils中改配置
        List<Person> list = getPageData2(currentPage, lineSize);
        for (Person person : list) {
            System.out.println(person);
        }

    }

    //使用JDBC连接Mysql数据库实现分页
    public static List<Person> getPageData(int currentPage, int lineSize)
throws SQLException {

        //从哪个位置开始取数据
        int start = (currentPage - 1) * lineSize;

        QueryRunner queryRunner = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "SELECT name,address  FROM person LIMIT ?,?";

        List<Person> persons = (List<Person>) queryRunner.query(sql, new
BeanListHandler(Person.class), new Object[]{start, lineSize});
        return persons;

    }

    //使用JDBC连接Oracle数据库实现分页
    public static List<Person> getPageData2(int currentPage, int lineSize)
throws SQLException {

        //从哪个位置开始取数据
        int start = (currentPage - 1) * lineSize;

        //读取前N条数据
        int end = currentPage * lineSize;

        QueryRunner queryRunner = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "SELECT " +
                "  name, " +
                "  address " +
                "FROM ( " +
                "  SELECT " +
                "    name, " +
                "    address , " +
                "    ROWNUM rn " +
                "  FROM person " +
```

```
                "  WHERE ROWNUM <= ? " +
                ")temp WHERE temp.rn>?";

        List<Person> persons = (List<Person>) queryRunner.query(sql, new
BeanListHandler(Person.class), new Object[]{end, start});
        return persons;

    }

    public static int getPageCount(int totalRecord, int lineSize) {

        //简单算法
        //return (totalRecord - 1) / lineSize + 1;

        //此算法比较好理解，把数据代代进去就知道了。
        return totalRecord % lineSize == 0 ? (totalRecord / lineSize) :
(totalRecord / lineSize) + 1;

    }


    public static int  getTotalRecord() throws SQLException {

        //使用DbUtils框架查询数据库表中有多少条数据
        QueryRunner queryRunner = new QueryRunner(JdbcUtils.getDataSource());
        String sql = "SELECT COUNT(*) FROM person";

        Object o = queryRunner.query(sql, new ScalarHandler());

        String ss = o.toString();
        int  s = Integer.parseInt(ss);
        return s;
    }
```

加油

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

# 面试题

## 1. JDBC操作数据库的步骤？

> JDBC操作数据库的步骤？

1. 注册数据库驱动。
2. 建立数据库连接。
3. 创建一个Statement。
4. 执行SQL语句。
5. 处理结果集。
6. 关闭数据库连接

代码如下：

```
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {

            /*
            * 加载驱动有两种方式
            *
            * 1：会导致驱动会注册两次，过度依赖于mysql的api，脱离的mysql的开发包，程序则
无法编译
            * 2：驱动只会加载一次，不需要依赖具体的驱动，灵活性高
            *
            * 我们一般都是使用第二种方式
            * */

            //1.
            //DriverManager.registerDriver(new com.mysql.jdbc.Driver());

            //2.
            Class.forName("com.mysql.jdbc.Driver");

            //获取与数据库连接的对象-Connetcion
            connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/zhongfucheng",
"root", "root");

            //获取执行sql语句的statement对象
            statement = connection.createStatement();

            //执行sql语句,拿到结果集
            resultSet = statement.executeQuery("SELECT * FROM users");

            //遍历结果集，得到数据
            while (resultSet.next()) {

                System.out.println(resultSet.getString(1));

                System.out.println(resultSet.getString(2));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } finally {

            /*
```

```
            * 关闭资源，后调用的先关闭
            *
            * 关闭之前，要判断对象是否存在
            * */

            if (resultSet != null) {
                try {
                    resultSet.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }

            }
            if (statement != null) {
                try {
                    statement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }

            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }

            }

        }
```

# 2. JDBC中的Statement 和PreparedStatement，CallableStatement的区别?

> JDBC中的Statement 和PreparedStatement的区别?

区别：

- PreparedStatement是预编译的SQL语句，效率高于Statement。
- PreparedStatement支持?操作符，相对于Statement更加灵活。
- PreparedStatement可以防止SQL注入，安全性高于Statement。
- CallableStatement适用于执行存储过程。

# 3. JDBC中大数据量的分页解决方法?

> JDBC中大数据量的分页解决方法?

最好的办法是利用sql语句进行分页，这样每次查询出的结果集中就只包含某页的数据内容。

mysql语法：

```
SELECT *
FROM 表名
LIMIT [START], length;
```

oracle语法：

```
SELECT *FROM (
    SELECT 列名,列名,ROWNUM rn
    FROM 表名
    WHERE ROWNUM<=(currentPage*lineSize)) temp

WHERE temp.rn>(currentPage-1)*lineSize;
```

# 4. 说说数据库连接池工作原理和实现方案?

> 说说数据库连接池工作原理和实现方案?

工作原理：

- 服务器启动时会建立一定数量的池连接，并一直维持不少于此数目的池连接。客户端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量有配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。

实现方案：**连接池使用集合来进行装载，返回的Connection是原始Connection的代理，代理Connection的close方法，当调用close方法时，不是真正关连接，而是把它代理的Connection对象放回到连接池中，等待下一次重复利用。**

具体代码：

```
@Override
   public Connection getConnection() throws SQLException {

       if (list.size() > 0) {
           final Connection connection = list.removeFirst();

           //看看池的大小
           System.out.println(list.size());

           //返回一个动态代理对象
```

```
            return (Connection)
Proxy.newProxyInstance(Demo1.class.getClassLoader(),
connection.getClass().getInterfaces(), new InvocationHandler() {

                @Override
                public Object invoke(Object proxy, Method method, Object[]
args) throws Throwable {

                    //如果不是调用close方法，就按照正常的来调用
                    if (!method.getName().equals("close")) {
                        method.invoke(connection, args);
                    } else {

                        //进到这里来，说明调用的是close方法
                        list.add(connection);

                        //再看看池的大小
                        System.out.println(list.size());

                    }
                    return null;
                }

            });
        }
        return null;
    }
```

# 5. Java中如何进行事务的处理?

> Java中如何进行事务的处理?

1. **事务是作为单个逻辑工作单元执行的一系列操作。**
2. 一个逻辑工作单元必须有四个属性，称为**原子性、一致性、隔离性和持久性 (ACID) 属性，只有这样才能成为一个事务**

Connection类中提供了4个事务处理方法:

- setAutoCommit(Boolean autoCommit):设置是否自动提交事务,默认为自动提交,即为true,通过设置false禁止自动提交事务;
- commit():提交事务;
- rollback():回滚事务.
- savepoint:保存点
    - 注意：savepoint不会结束当前事务，普通提交和回滚都会结束当前事务的

# 6. 修改JDBC代码质量

> 下述程序是一段简单的基于JDBC的数据库访问代码,实现了以下功能:从数据库中查询product表中的所有记录,然后打印输出到控制台.该代码质量较低,如没有正确处理异常,连接字符串以"魔数"的形式直接存在于代码中等,请用你的思路重新编写程序,**完成相同的功能,提高代码质量.**

原来的代码:

```
public void printProducts(){
    Connection c = null;
    Statements s = null;
    ResultSet r = null;
    try{

c=DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:sid","username
","password");
        s=c.createStatement();
        r=s.executeQuery("select id, name, price from product");
        System.out.println("Id\tName\tPrice");
        while(r.next()){
            int x = r.getInt("id");
            String y = r.getString("name");
            float z = r.getFloat("price");
            System.out.println(x + "\t" + y + "\t" + z);
        }
    } catch(Exception e){

    }
}
```

修改后的代码:

```
class Constant{
    public static final String URL="jdbc:oracle:thin:@127.0.0.1:1521:sid";
    public static final String USERNAME="username";
    public static final String PASSWORD="password";
}

class DAOException extends Exception{
    public DAOException(){
        super();
    }
    public DAOException(String msg){
        super(msg);
    }
}

public class Test{
```

```
    public void printProducts() throws DAOException{
        Connection c = null;
        Statement s = null;
        ResultSet r = null;
        try{
            c =
DriverManager.getConnection(Constant.URL,Constant.USERNAME,Constant.PASSWORD);
            s = c.createStatement();
            r = s.executeQuery("select id,name,price from product");
            System.out.println("Id\tName\tPrice");
            while(r.next()){
                int x = r.getInt("id");
                String y = r.getString("name");
                float z = r.getFloat("price");
                System.out.println(x + "\t" + y + "\t" + z);
            }
        } catch (SQLException e){
            throw new DAOException("数据库异常");
        } finally {
            try{
                r.close();
                s.close();
                c.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

修改点：

- url、password等信息不应该直接使用字符串“写死”，可以使用常量代替
- catch中应该回滚事务，抛出RuntimeException也是回滚事务的一种方法
- 关闭资源

# 7. 写出一段JDBC连接本机MySQL数据库的代码

> 写出一段JDBC连接本机MySQL数据库的代码

```
Class.forName("com.mysql.jdbc.Driver");
String url="jdbc:mysql://localhost/test";
Stirng user='root';
String password='root';
Connection conn = DriverManager.getConnection(url,user,password);
```

# 8. JDBC是如何实现Java程序和JDBC驱动的松耦合的?

> JDBC是如何实现Java程序和JDBC驱动的松耦合的?

通过制定接口，数据库厂商来实现。我们只要通过接口调用即可。随便看一个简单的JDBC示例，**你会发现所有操作都是通过JDBC接口完成的，而驱动只有在通过Class.forName反射机制来加载的时候才会出现。**

# 9. execute，executeQuery，executeUpdate的区别是什么?

> execute，executeQuery，executeUpdate的区别是什么?

- Statement的execute(String query)方法用来**执行任意的SQL查询**，如果查询的结果是一个ResultSet，这个方法就返回true。如果结果不是ResultSet，比如insert或者update查询，它就会返回false。我们可以通过它的getResultSet方法来获取ResultSet，或者通过getUpdateCount()方法来获取更新的记录条数。
- Statement的executeQuery(String query)接口用来执行select查询，并且返回ResultSet。即使查询不到记录返回的ResultSet也不会为null。我们通常使用executeQuery**来执行查询语句**，这样的话如果传进来的是insert或者update语句的话，它会抛出错误信息为 “executeQuery method can not be used for update”的java.util.SQLException。
- Statement的executeUpdate(String query)方法**用来执行insert或者update/delete（DML）语句，或者 什么也不返回DDL语句**。返回值是int类型，如果是DML语句的话，它就是更新的条数，如果是DDL的话，就返回0。
- 只有当你不确定是什么语句的时候才应该使用execute()方法，否则应该使用executeQuery或者executeUpdate方法。

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

# 10. PreparedStatement的缺点是什么，怎么解决这个问题?

> PreparedStatement的缺点是什么，怎么解决这个问题?

PreparedStatement的一个缺点是，**我们不能直接用它来执行in条件语句**；需要执行IN条件语句的话，下面有一些解决方案：

- 分别进行单条查询——这样做性能很差，不推荐。
- 使用存储过程——这取决于数据库的实现，不是所有数据库都支持。
- 动态生成PreparedStatement——这是个好办法，但是不能享受PreparedStatement的缓存带来的好处了。
- 在PreparedStatement查询中使用NULL值——如果你知道输入变量的最大个数的话，这是个不错的办法，扩展一下还可以支持无限参数。

# 11. JDBC的脏读是什么？哪种数据库隔离级别能防止脏读?

> JDBC的脏读是什么？哪种数据库隔离级别能防止脏读?

脏读：**一个事务读取到另外一个事务未提交的数据**

例子：A向B转账，A执行了转账语句，但A还没有提交事务，B读取数据，发现自己账户钱变多了！B跟A说，我已经收到钱了。A回滚事务【rollback】，等B再查看账户的钱时，发现钱并没有多。

下面的三种个隔离级别都可以防止：

- Serializable【TRANSACTION_SERIALIZABLE】
- Repeatable read【TRANSACTION_REPEATABLE_READ】

- Read committed【TRANSACTION_READ_COMMITTED】

# 12. 什么是幻读，哪种隔离级别可以防止幻读?

> 什么是幻读，哪种隔离级别可以防止幻读?

**是指在一个事务内读取到了别的事务插入的数据，导致前后读取不一致。**

**只有TRANSACTION_SERIALIZABLE隔离级别才能防止产生幻读。**

# 13. JDBC的DriverManager是用来做什么的?

> JDBC的DriverManager是用来做什么的?

- JDBC的DriverManager是一个**工厂类**，我们**通过它来创建数据库连接。**
- 当JDBC的Driver类被加载进来时，它会自己注册到DriverManager类里面
- 然后我们会把数据库配置信息传成DriverManager.getConnection()方法，**DriverManager会使用注册到它里面的驱动来获取数据库连接，并返回给调用的程序。**

# 14. JDBC的ResultSet是什么?

> JDBC的ResultSet是什么?

- **在查询数据库后会返回一个ResultSet，它就像是查询结果集的一张数据表。**
- ResultSet对象维护了一个游标，指向当前的数据行。开始的时候这个游标指向的是第一行。如果调用了ResultSet的next()方法游标会下移一行，如果没有更多的数据了，next()方法会返回false。可以在for循环中用它来遍历数据集。
- 默认的ResultSet是不能更新的，游标也只能往下移。也就是说你只能从第一行到最后一行遍历一遍。不过也可以创建可以回滚或者可更新的ResultSet
- **当生成ResultSet的Statement对象要关闭或者重新执行或是获取下一个ResultSet的时候，ResultSet对象也会自动关闭。**
- 可以通过ResultSet的getter方法，传入列名或者从1开始的序号来获取列数据。

# 15. 有哪些不同的ResultSet?

> 有哪些不同的ResultSet?

根据创建Statement时输入参数的不同，会对应不同类型的ResultSet。如果你看下Connection的方法，你会发现createStatement和prepareStatement方法重载了，以支持不同的ResultSet和并发类型。

一共有三种ResultSet对象。

- **ResultSet.TYPE_FORWARD_ONLY：这是默认的类型，它的游标只能往下移。**
- **ResultSet.TYPE_SCROLL_INSENSITIVE：游标可以上下移动，一旦它创建后，数据库里的数据再发生修改，对它来说是透明的。**
- **ResultSet.TYPE_SCROLL_SENSITIVE：游标可以上下移动，如果生成后数据库还发生了修改操作，它是能够感知到的。**

ResultSet有两种并发类型。

- ResultSet.CONCUR_READ_ONLY:**ResultSet是只读的，这是默认类型。**
- ResultSet.CONCUR_UPDATABLE:**我们可以使用ResultSet的更新方法来更新里面的数据。**

# 16. JDBC的DataSource是什么，有什么好处

> JDBC的DataSource是什么，有什么好处

DataSource即数据源，它是定义在javax.sql中的一个接口，跟DriverManager相比，它的功能要更强大。**我们可以用它来创建数据库连接，当然驱动的实现类会实际去完成这个工作。除了能创建连接外，它还提供了如下的特性：**

- **缓存PreparedStatement以便更快的执行**
- **可以设置连接超时时间**
- **提供日志记录的功能**
- **ResultSet大小的最大阈值设置**
- **通过JNDI的支持，可以为servlet容器提供连接池的功能**

# 17 .如何通过JDBC的DataSource和Apache Tomcat的JNDI来创建连接池?

> 如何通过JDBC的DataSource和Apache Tomcat的JNDI来创建连接池?

Tomcat服务器也给我们提供了连接池，内部其实就是DBCP

步骤：

1. 在META-INF目录下配置context.xml文件【文件内容可以在tomcat默认页面的 JNDI Resources下Configure Tomcat's Resource Factory找到】
2. 导入Mysql或oracle开发包到tomcat的lib目录下
3. 初始化JNDI->获取JNDI容器->检索以XXX为名字在JNDI容器存放的连接池

context.xml文件的配置：

```
<Context>

  <Resource name="jdbc/EmployeeDB"
            auth="Container"
            type="javax.sql.DataSource"

            username="root"
            password="root"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/zhongfucheng"
            maxActive="8"
            maxIdle="4"/>
</Context>
```

```
        try {

      //初始化JNDI容器
            Context initCtx = new InitialContext();

      //获取到JNDI容器
            Context envCtx = (Context) initCtx.lookup("java:comp/env");

      //扫描以jdbc/EmployeeDB名字绑定在JNDI容器下的连接池
            DataSource ds = (DataSource)
                    envCtx.lookup("jdbc/EmployeeDB");

            Connection conn = ds.getConnection();
            System.out.println(conn);

        }
```

# 18. Apache的DBCP是什么?

> Apache的DBCP是什么

如果用DataSource来获取连接的话，**通常获取连接的代码和驱动特定的DataSource是紧耦合的**。另外，除了选择DataSource的实现类，剩下的代码基本都是一样的。

Apache的DBCP就是用来解决这些问题的，它提供的DataSource实现成为了应用程序和不同JDBC驱动间的一个抽象层。**Apache的DBCP库依赖commons-pool库，所以要确保它们都在部署路径下。**

使用DBCP数据源的步骤：

1. 导入两个jar包【Commons-dbcp.jar和Commons-pool.jar】
2. 读取配置文件
3. 获取BasicDataSourceFactory对象
4. 创建DataSource对象

```
    private static DataSource dataSource = null;

    static {
        try {
            //读取配置文件
            InputStream inputStream =
Demo3.class.getClassLoader().getResourceAsStream("dbcpconfig.properties");
            Properties properties = new Properties();
            properties.load(inputStream);
```

```
            //获取工厂对象
            BasicDataSourceFactory basicDataSourceFactory = new
BasicDataSourceFactory();
            dataSource = basicDataSourceFactory.createDataSource(properties);

        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection() throws SQLException {
        return dataSource.getConnection();

    }

    //这里释放资源不是把数据库的物理连接释放了，是把连接归还给连接池【连接池的Connection内
部自己做好了】
    public static void release(Connection conn, Statement st, ResultSet rs) {

        if (rs != null) {
            try {
                rs.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            rs = null;
        }
        if (st != null) {
            try {
                st.close();
            } catch (Exception e) {
                e.printStackTrace();
            }

        }
        if (conn != null) {
            try {
                conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }

        }


    }
```

# 19. 常见的JDBC异常有哪些?

> 常见的JDBC异常有哪些?

有以下这些：

- java.sql.SQLException——这是JDBC异常的基类。
- java.sql.BatchUpdateException——当批处理操作执行失败的时候可能会抛出这个异常。这取决于具体的JDBC驱动的实现，它也可能直接抛出基类异常java.sql.SQLException。
- java.sql.SQLWarning——SQL操作出现的警告信息。
- java.sql.DataTruncation——字段值由于某些非正常原因被截断了（不是因为超过对应字段类型的长度限制）。


如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y

# 20. JDBC中存在哪些不同类型的锁?

> JDBC中存在哪些不同类型的锁?

从广义上讲，有两种锁机制来防止多个用户同时操作引起的数据损坏。

- **乐观锁——只有当更新数据的时候才会锁定记录。**
- **悲观锁——从查询到更新和提交整个过程都会对数据记录进行加锁。**

# 21. java.util.Date和java.sql.Date有什么区别?

> java.util.Date和java.sql.Date有什么区别?

java.util.Date包含日期和时间，而java.sql.Date只包含日期信息，而没有具体的时间信息。**如果你想把时间信息存储在数据库里，可以考虑使用Timestamp或者DateTime字段**

# 22. SQLWarning是什么，在程序中如何获取SQLWarning?

> SQLWarning是什么，在程序中如何获取SQLWarning?

SQLWarning是SQLException的子类，**通过Connection, Statement, Result的getWarnings方法都可以获取到它**。 SQLWarning不会中断查询语句的执行，只是用来提示用户存在相关的警告信息。

# 23. 如果java.sql.SQLException: No suitable driver found该怎么办?

> 如果java.sql.SQLException: No suitable driver found该怎么办?

如果你的SQL URL串格式不正确的话，就会抛出这样的异常。不管是使用DriverManager还是JNDI数据源来创建连接都有可能抛出这种异常。它的异常栈看起来会像下面这样。

```
org.apache.tomcat.dbcp.dbcp.SQLNestedException: Cannot create JDBC driver of
class 'com.mysql.jdbc.Driver' for connect URL
''jdbc:mysql://localhost:3306/UserDB'
    at
org.apache.tomcat.dbcp.dbcp.BasicDataSource.createConnectionFactory(BasicDataS
ource.java:1452)
    at
org.apache.tomcat.dbcp.dbcp.BasicDataSource.createDataSource(BasicDataSource.j
ava:1371)
    at
org.apache.tomcat.dbcp.dbcp.BasicDataSource.getConnection(BasicDataSource.java
:1044)
java.sql.SQLException: No suitable driver found for
'jdbc:mysql://localhost:3306/UserDB
    at java.sql.DriverManager.getConnection(DriverManager.java:604)
    at java.sql.DriverManager.getConnection(DriverManager.java:221)
    at com.journaldev.jdbc.DBConnection.getConnection(DBConnection.java:24)
    at com.journaldev.jdbc.DBConnectionTest.main(DBConnectionTest.java:15)
Exception in thread "main" java.lang.NullPointerException
    at com.journaldev.jdbc.DBConnectionTest.main(DBConnectionTest.java:16)
```

解决这类问题的方法就是，检查下日志文件，像上面的这个日志中，URL串是'jdbc:mysql://localhost:3306/UserDB，只要把它改成jdbc:mysql://localhost:3306/UserDB就好了。

# 24. JDBC的RowSet是什么，有哪些不同的RowSet?

> JDBC的RowSet是什么，有哪些不同的RowSet?

RowSet用于存储查询的数据结果，和ResultSet相比，**它更具灵活性。RowSet继承自ResultSet，因此ResultSet能干的，它们也能，而ResultSet做不到的，它们还是可以**。RowSet接口定义在javax.sql包里。

RowSet提供的额外的特性有：

- 提供了Java Bean的功能，可以通过settter和getter方法来设置和获取属性。RowSet使用了JavaBean的事件驱动模型，它可以给注册的组件发送事件通知，比如游标的移动，行的增删改，以及RowSet内容的修改等。
- RowSet对象默认是**可滚动，可更新的，因此如果数据库系统不支持ResultSet实现类似的功能，可以使用RowSet来实现**。

RowSet分为两大类：

- A. **连接型RowSet——这类对象与数据库进行连接**，和ResultSet很类似。JDBC接口只提供了一种连接型RowSet，javax.sql.rowset.JdbcRowSet，它的标准实现是com.sun.rowset.JdbcRowSetImpl。
- B. **离线型RowSet——这类对象不需要和数据库进行连接，因此它们更轻量级，更容易序列化。它们适用于在网络间传递数据**。

- 有四种不同的离线型RowSet的实现。
  - CachedRowSet——可以通过他们获取连接，执行查询并读取ResultSet的数据到RowSet里。我们可以在离线时对数据进行维护和更新，然后重新连接到数据库里，并回写改动的数据。
  - WebRowSet继承自CachedRowSet——他可以读写XML文档。
  - JoinRowSet继承自WebRowSet——它不用连接数据库就可以执行SQL的join操作。
  - FilteredRowSet继承自WebRowSet——我们可以用它来设置过滤规则，这样只有选中的数据才可见。

# 25. 什么是JDBC的最佳实践?

> 什么是JDBC的最佳实践?

- 数据库资源是非常昂贵的，**用完了应该尽快关闭它**。Connection, Statement, ResultSet等JDBC对象都有close方法，调用它就好了。
- **养成在代码中显式关闭掉ResultSet，Statement，Connection的习惯**，如果你用的是连接池的话，连接用完后会放回池里，但是没有关闭的ResultSet和Statement就会造成资源泄漏了。
- 在finally块中关闭资源，保证即便出了异常也能正常关闭。
- **大量类似的查询应当使用批处理完成**。
- **尽量使用PreparedStatement而不是Statement，以避免SQL注入，同时还能通过预编译和缓存机制提升执行的效率。**
- 如果你要将大量数据读入到ResultSet中，**应该合理的设置fetchSize以便提升性能**。
- 你用的数据库可能没有支持所有的隔离级别，用之前先仔细确认下。
- 数据库隔离级别越高性能越差，确保你的数据库连接设置的隔离级别是最优的。
- 如果在WEB程序中创建数据库连接，最好通过JNDI使用JDBC的数据源，这样可以对连接进行重用。
- **如果你需要长时间对ResultSet进行操作的话，尽量使用离线的RowSet。**

如果文档中有任何的不懂的问题，都可以直接来找我询问，我乐意帮助你们！微信搜**Java3y**公众号有我的联系方式。更多**原创**技术文章可关注我的GitHub：https://github.com/ZhongFuCheng3y/3y